TRABAJO DE FIN DE GRADO:

Desarrollo de herramienta para comunicación con vehículo a través de CAN-bus.

Autor:

Miguel Ángel de Miguel Paraíso.

Grado en ingeniería electrónica industrial y automática.

Tutor del proyecto:

Fernando García Fernández.

Director del proyecto

Juan Carmona Fernández.

Departamento de ingeniería de sistemas y automática.

Universidad Carlos III de Madrid.

Junio 2015.

Agradecimientos

A mis padres,

a los integrantes del laboratorio LSI.

Índice General

Índice de Ilustraciones

Índice de tablas

Abreviaturas

CAN: Controlled Area Network .

ISO: International Organization for Standardization (Organización Internacional de Estandarización).

SIMBA: Sistema Integrado de Monitorización Bidireccional del Automóvil.

IVVI: Intelligent Vehicle based on Visual Information (Vehículo Inteligente basado en Información Visual).

LSI: Laboratorio de Sistemas Inteligentes.

ADAS: Advanced Driver Assistance Systems (Sistemas avanzados de asistencia al conductor).

ROS: Robot Operating System (Sistema Operativo para Robot).

RPi: Raspberry Pi.

1. Introducción

Durante los últimos años, los automóviles han ido incorporando cada vez más componentes electrónicos entre sus elementos.

Estos sistemas electrónicos, se empezaron a implantar por primera vez para controlar diferentes parámetros de la combustión del motor y poder mejorar así su eficiencia.

A partir de ahí, los fabricantes han ido introduciendo electrónica en el resto de elementos del vehículo, para poder, tanto controlarlos, como monitorizarlos, hasta tal punto que hoy se han vuelto imprescindibles en el diseño de cualquier automóvil.

Las funciones de los sistemas electrónicos, van desde la monitorización de sensores que pueden detectar posibles accidentes y evitarlos (ABS o ESP), hasta los sistemas de confort de los pasajeros y conductor (elevalunas eléctrico, radio, etc).

Para administrar todos estos dispositivos electrónicos se necesitan principalmente: una computadora central que se encargue de coordinar y hacer funcionar todos los elementos correctamente, y una red de comunicación que conecte todos los elementos con la computadora central. Dicha red sigue un estándar en prácticamente la totalidad de los automóviles y está definida por una serie de normas ISO.

Esta red de comunicaciones llamada bus CAN es sobre la que se va a desarrollar todo el proyecto.

Por ella circula información esencial de los elementos más importantes del vehículo, como la dirección, el acelerador, el freno, etc.

Leer estos mensajes que circulan por el bus CAN, puede ser de gran utilidad para diagnosticar problemas en algún elemento del vehículo, o probar piezas simulando que están en el vehículo.

Además, dichos mensajes pueden aportar información sobre parámetros del vehículo, como la velocidad en cada rueda, el ángulo de giro de la dirección, o si se ha activado el freno. Esta información puede ser utilizada para desarrollar

nuevos sistemas en los vehículos, como detección del perfil de conductor, ayuda a la localización mediante la odometría del vehículo, etc.

Por otro lado, enviando mensajes por el bus CAN, es posible simular los datos que envía un elemento, el acelerador por ejemplo y controlarlo remotamente desde un ordenador para controlar la velocidad del vehículo.

1.1. Objetivos

El objetivo principal de este trabajo, es realizar un dispositivo de bajo coste que sea capaz de leer y escribir en el bus CAN de un automóvil.

Dicho dispositivo tendrá que acceder al bus CAN a través del puerto OBDII, un puerto universal que se encuentra en la mayoría de los vehículos y se utiliza para tareas de diagnosis y por el que se puede acceder al bus CAN.

El sistema tendrá que ser capaz de soportar una velocidad en el bus CAN de hasta 1Mbps, que es la máxima que se puede encontrar en un vehículo.

Además, el dispositivo final debe ser un sistema portable y compacto para que sea fácil de transportar y pueda funcionar en el vehículo sin necesidad de conectar más elementos.

Para realizar esto, esnecesario:

- Elegir el hardware, valorando distintas alternativas hasta llegar a la más adecuada en relación funcionalidad/coste.
- Diseñar el software que permita leer y escribir mensajes en el CAN bus, creando una librería que posibilite además añadir nuevas funcionalidades en el futuro de forma fácil si fuera necesario.
- Crear una interfaz gráfica que permita realizar las tareas de lectura y escritura de forma rápida y cómoda.
- Diseñar una cubierta donde se puedan introducir todos los elementos, quedando como resultado final un sistema de pequeño tamaño y portable.

Al sistema desarrollado se le dará el nombre de "*Sistema Integrado de Monitorización Bidireccional del Automóvil*" y se referirá a él a través de su acrónimo SIMBA.

1.2. Plataforma de pruebas

Para realizar pruebas con SIMBA, se ha trabajado sobre un vehículo real: el IVVI 2.0.

IVVI (*Intelligent Vehicle based on Visual Information*) 2.0 es la segunda plataforma de investigación del LSI para la implementación de sistemas inteligentes basados en visión por computador y técnicas de láser, con el propósito de desarrollar y probar tecnologías ADAS. [1]

Ilustración 1 - IVVI 2.0

El sistema de procesamiento de datos del vehículo se encuentra en el maletero y consta principalmente de un ordenador al que se le ha instalado ROS para adquirir y procesar los datos de los distintos sensores del vehículo.

En este proyecto se ha utilizado el IVVI 2.0 como banco de pruebas, tanto tomando los datos que se transmiten por su bus CAN, como enviando mensajes para controlar algunas de sus funciones.

2. ¿Qué es el bus CAN?

El bus CAN (Controller Area Network) es un bus para vehículos estándar, diseñado para permitir la comunicación entre los distintos elementos de un vehículo.

Este protocolo fue desarrollado por la compañía BOSCH en 1983, pensando en crear una red multiplexada para la comunicación entre unidades en un ambiente distribuido.

En los vehículos existen una gran cantidad de elementos que necesitan enviar o recibir información de otros.

Una comunicación punto a punto supondría una gran cantidad de cables que aumentarían el peso y la complejidad del sistema, además restarían volumen que se puede aprovechar para otras funciones.

Es por esto que se necesita una red multiplexada, es decir, una red con un circuito único que intercomunique todos los elementos con un cableado de tipo bus.

Todos los dispositivos electrónicos que estén conectados al bus CAN, recibirán todos los mensajes que publiquen otros dispositivos, y a su vez, todos los mensajes que envíe podrán ser leídos por cualquier elemento conectado al bus.

Otra de las ventajas de este protocolo, reside en que el procesador principal delega la carga de comunicaciones a un periférico independiente que las controla. De esta manera, el procesador principal puede dedicar más recursos a realizar sus tareas.

Además, este protocolo está normalizado, de tal forma que se reducen costes a la hora de comunicar elementos de distintos fabricantes en la misma red.

Las características que debe tener el bus CAN de alta velocidad vienen descritas en la norma **ISO 11898**.

Algunas de estas características vienen descritas a continuación.

2.1. Cables.

El medio físico por el que se transmite la señal consta de dos cables: CAN High (o CANH) y CAN Low (o CANL). La señal que se transmite por este par de cables es diferencial, es decir, es necesario restar el valor de tensión del CANL al CANH para conocer la señal del bit.

Para reducir las interferencias, estos dos cables van trenzados, de tal forma que las interferencias les afecten a los dos por igual. Como para conocer la señal del bit hay que restar las dos señales, la interferencia se anula.

Para que la señal se propague correctamente por el bus, éste debe cumplir unas especificaciones de resistencia: la resistencia de la red debe ser de 120 +-18 Ohm.

Dado que la señal no se propaga inmediatamente, sino que tiene un cierto retardo, la longitud del bus está limitada y depende de la velocidad de transmisión tal y como se muestra en la Tabla 1.

Velocidad del bus	Tiempo del bit	Longitud del bus
1Mbit/Sec	1 µs	30 metros
500Kb/Sec	2 µs	100 metros
250Kb/Sec	4 µs	250 metros
125Kb/Sec	8 µs	1000 metros
50Kb/Sec	20 µs	500 metros
20Kb/Sec	50 µs	2500 metros
10Kb/Sec	100 µs	5000 metros

Tabla 1 - Velocidad del bus

2.2. Nivel de señal

La señal que codifica cada bit de información, se obtiene restando el nivel del CANL al CANH, por lo tanto, cada cable tiene asociado un nivel distinto para el '0' y para el '1'.

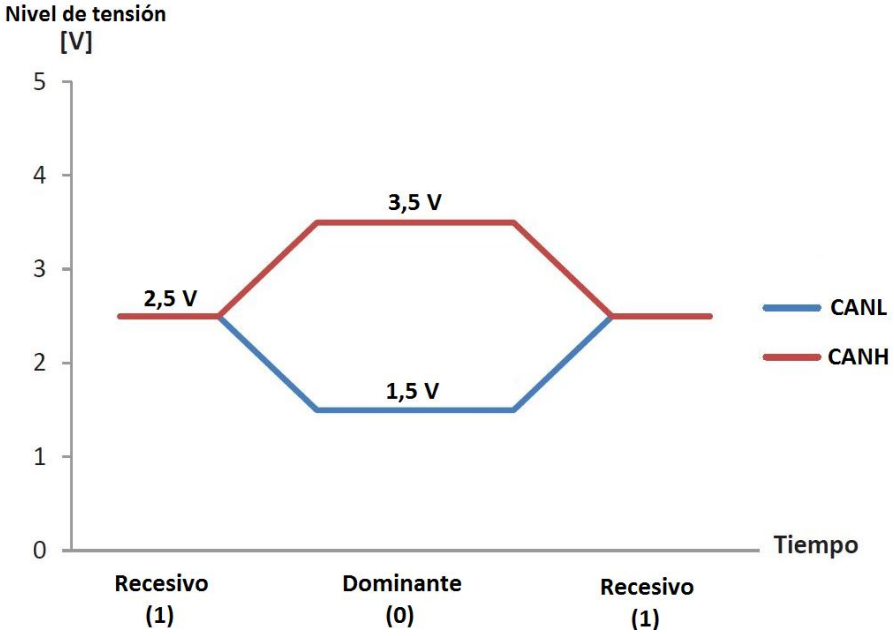


Ilustración 2 - Nivel de tensión

Para un '1', los niveles son de 3,5V en CANH y 1,5V en CANL.

Para un '0', los niveles son de 2,5V en CANH y en CANL.

De esta manera, al restar CANH – CANL, el nivel para el '1' es de 2V y para el cero es de 0 Voltios.

2.3. Prioridades.

En el sistema de codificación de este protocolo, el bit dominante es el '0'. Esto quiere decir que en caso de que dos nodos distintos envíen un uno y un cero a la vez, el cero dominará y el uno no será enviado (se forzará a un cero).

Por lo tanto, si en un momento dado dos elementos electrónicos se disponen a enviar a la vez un mensaje, predominará el que esté enviando un cero y el otro tendrá que esperar para enviar su mensaje que es menos prioritario.

De esta manera, como todos los mensajes empiezan por la cabecera, serán más prioritarios los que tengan un valor menor en este campo.

2.4. Tiempo nominal de Bit.

A la hora de transmitir los bits por los cables del bus, hay que tener en cuenta una serie de retardos en la transmisión de cada bit, que indican el momento en el que hay que muestrear el bit para tener la certeza de que está estable y todavía no se está propagando por el bus.

El tiempo en el que se está transmitiendo el bit se divide en diferentes segmentos:

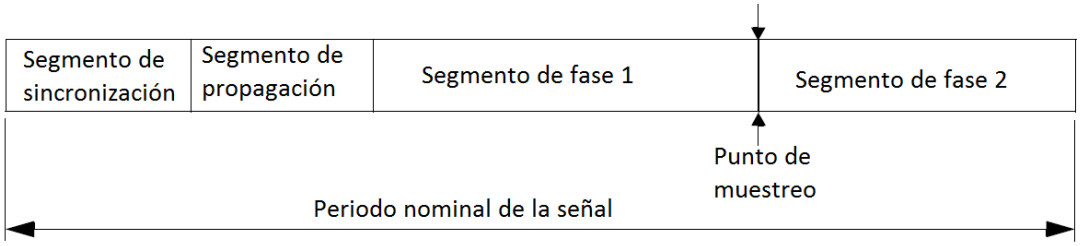


Ilustración 3 - Tiempo de bit

El segmento de sincronización determina el tiempo que tardan los nodos en sincronizarse desde que se envía el bit.

El segmento de propagación recoge el tiempo que tarda la señal en propagarse por todo el bus y tiene en cuenta los retardos que pueda haber por rebotes de la señal.

Por último, los segmentos de fase 1 y 2 determinan el punto de muestreo, que se encuentra entre ellos. El punto de muestreo se puede modificar acortando un segmento y alargando otro ya que los tiempos de sincronización y propagación suelen ser fijos.

2.5. Estructura del mensaje.

Los mensajes que se envían por el bus CAN tienen todos la misma estructura. La información está almacenada en distintos campos del mensaje, existiendo cinco tipos de mensajes:

2.5.1. Mensaje CAN estándar:

Los mensaje de tipo estándar, son los más comunes en la comunicación interna de los vehículos. Constan de los siguientes campos:

- Cabecera o "Arbitration Field". Consta de 11 bits de información y es el que identifica el mensaje. Con la información de este campo, los componentes electrónicos deciden si el mensaje va dirigido a ellos o no.
- Campo de información del tipo de mensaje. Incluye información como el número de Bytes que va a tener el mensaje o el tipo de mensaje que es (estándar, extendido o remoto).
- Campo de datos. Es donde se encuentra la información que se quiere transmitir. Su longitud puede variar desde 1 Byte hasta 8.
- Campo de detección de errores o CRC (Cyclic Redundancy check). Sirve para comprobar si el mensaje se ha transmitido correctamente. Si se realiza una serie de operaciones matemáticas establecidas al resto del mensaje, el resultado debe coincidir con el valor de este campo si no ha habido ningún fallo en la transmisión.

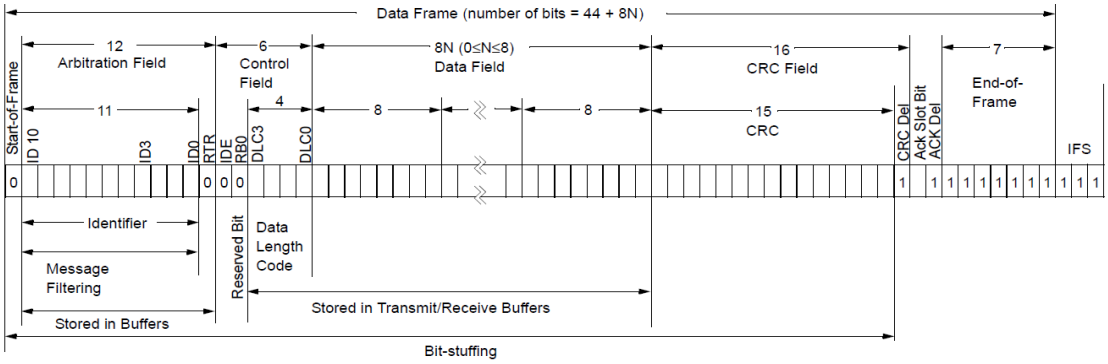


Ilustración 4 - Mensaje CAN estándar

2.5.2. Mensaje CAN extendido:

Los mensaje de CAN extendido, pueden transmitir el mismo número de Bytes que el anterior (hasta 8), sin embargo, el número de bits de la cabecera ("Arbitration Field") es de 29 (en lugar de 11 como en el mensaje anterior).

Su estructura es casi igual que la del mensaje CAN estándar. Las únicas modificaciones son los 18 nuevos bits de la cabecera, que se incluyen a continuación del bit IDE, y el valor de este bit (IDE), que pasa de ser '0' a ser '1', indicando que se trata de un mensaje extendido.

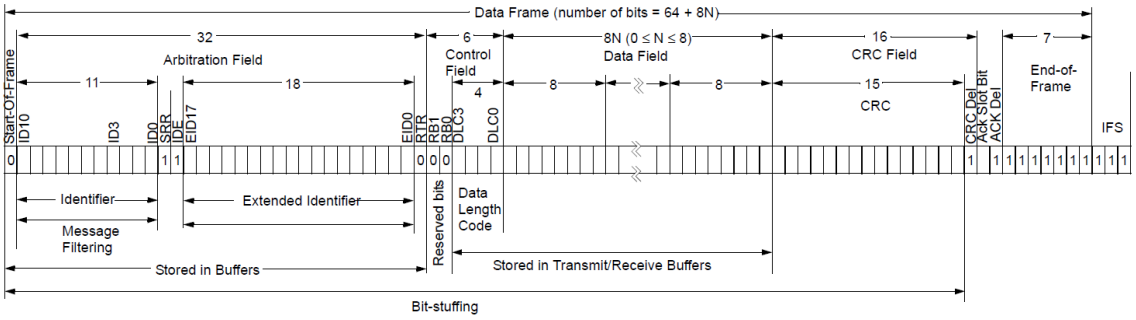


Ilustración 5 - Mensaje CAN extendido

2.5.3. Mensaje CAN remoto:

Normalmente, los elementos del vehículo envían mensajes con sus datos (por ejemplo los de un sensor), sin embargo, es posible también que un elemento pida los datos que necesite enviando un mensaje de este tipo.

Las diferencias con el mensaje estándar residen en que el bit RTR pasa de '0' a '1' y en que el campo de datos no existe, ya que no está enviando nada, solo está pidiendo información a un nodo.

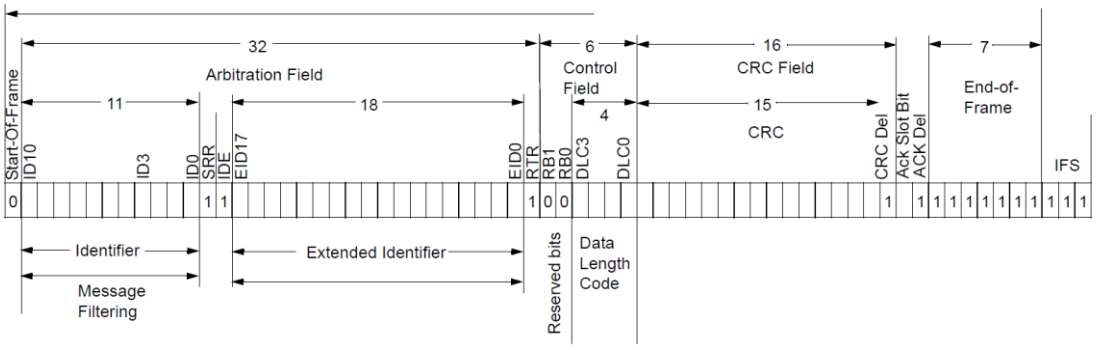


Ilustración 6 - Mensaje CAN remoto

2.5.4. Mensaje de error:

Los mensajes de error tienen dos campos. El primero son los flags de error (de 6 a 12 bits) que indican si el error es activo (todos los bits dominantes) o pasivo (todos los bits recesivos). El segundo es el delimitador, que consta de 8 bits recesivos ('1').

Este tipo de mensaje sólo se puede mandar mientras se está enviando un mensaje para indicar que hay algún error.

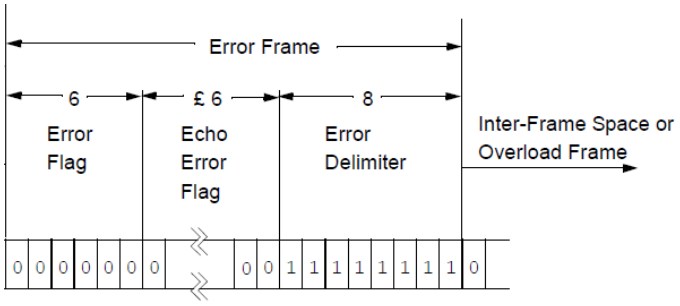


Ilustración 7 - Mensaje CAN de error

2.5.5. Mensaje de sobrecarga:

Los mensajes de sobrecarga tiene el mismo formato que un mensaje de error, pero se diferencia en que sólo se puede enviar mientras se está enviando el campo "inter-frame", nunca mientras se está enviando otro campo. Consta de 6 bits recesivos, seguidos de 8 bits dominantes.

Este tipo de mensaje, se puede enviar porque se detecta un bit dominante en el campo "inter-frame", que deben ser todos recesivos, o porque un nodo todavía no está preparado para enviar o recibir mensajes.

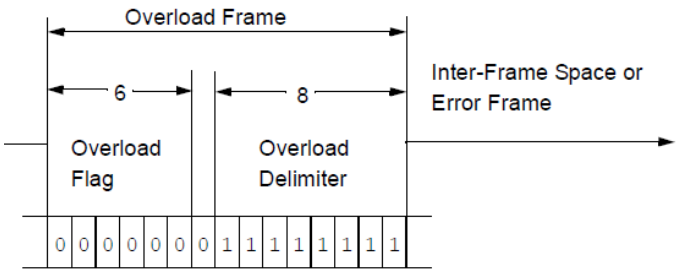


Ilustración 8 - Mensaje CAN de sobrecarga

3. Estado del arte

En la comunicación entre los distintos elementos del vehículo, existen distintos protocolos como KL-Line y SAE J1850, que se utilizan en los vehículos además del bus CAN.

Sin embargo, en los últimos años están surgiendo nuevas necesidades en la comunicación, ya que es necesario transmitir datos de video o audio y para ello se necesita más velocidad.

Es por esto, que los fabricantes han empezado a incluir nuevos protocolos para poder enviar estos datos.

3.1. Nuevos protocolos

A continuación, se describirán algunos de los nuevos protocolos de comunicación que se están desarrollando en torno a nuevas tecnologías como la fibra óptica.

3.1.1. MOST (Media Orientated Systems Transport).

Es una red multimedia de alta velocidad optimizada para los automóviles. Entre sus características destaca:

- La utilización de una topología de anillo y una transmisión de datos síncrona para transportar vídeo, audio, voz, etc.
- El medio de transmisión puede ser un conductor metálico o fibra óptica, con la que puede alcanzar velocidades de hasta 23 MegaBaudios.
- Se utiliza a nivel mundial en muchas de las marcas de coche más conocidas (Audi, BMW, Toyota, Seat, …).
- Un alto coste, que hace que sólo se incorpore en los coches más modernos que requieren la transmisión de audio y vídeo de alta calidad.

3.1.2. FlexRay

Es un protocolo desarrollado por un consorcio de diferentes empresas (entre los que se encuentran Bosch y algunos fabricantes de coches como BMW o General motors). Algunas de sus características son:

- Es un protocolo que está estandarizado y se puede utilizar en cualquier marca de coche. Sus características están definidas en las norma ISO 17458-1 a ISO 17458-5.
- El medio de transmisión puede ser un conductor de cobre o fibra óptica.
- Soporta velocidades de hasta 10Mbps y topologías tanto de bus como de estrella.
- Está diseñado para ser más rápido y más seguro que su antecesor CAN, sin embargo implantar una red FlexRay es mucho más caro que una CAN.

Aunque estos protocolos ofrecen mayor velocidad, en la industria del automóvil se siguen usando el bus CAN, KL-Line y SAE J1850 por su fiabilidad y bajo coste.

FlexRay y MOST, si se incluyen, actúan de complemento para transmitir un tipo de datos determinado, ya que aunque son más rápidos, su coste es mucho mayor.

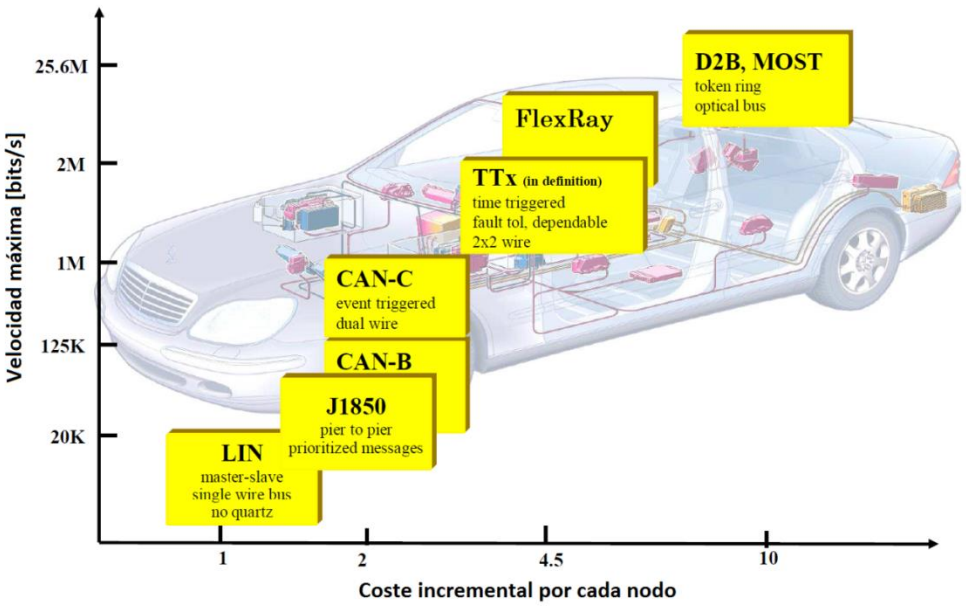


Ilustración 9 - Velocidad vs coste

En la Ilustración 9 se representa gráficamente la velocidad máxima de transmisión de datos y el coste de los distintos protocolos que puede haber en un vehículo.

3.2. Hardware para acceder al bus CAN

Actualmente, el bus CAN es el protocolo más extendido en el sector automovilístico y es por el que circulan mensajes con los parámetros más importantes.

El acceso al bus CAN es muy utilizado para tareas principalmente de diagnosis, ya que leyendo los mensajes que se transmiten se pueden conocer parámetros de los componentes del vehículo, y además se pueden enviar peticiones de datos específicas, para saber el estado de un componente en concreto.

Aunque existen algunos mensajes de diagnosis estándar, la mayoría son específicos de cada fabricante, y en general no existe una base de datos explicando el significado de cada mensaje.

Actualmente existen dos grupos de productos relacionados con el bus CAN, atendiendo a la tarea a las tareas que realizan:

- Los equipos de diagnosis, que leen los mensajes de diagnosis estándar y alguno específico del modelo del coche, para conocer su estado o el problema que tiene.
- Los equipos de lectura y/o escritura, que son capaces de leer todos los mensajes, pero no tienen una base de datos, por lo que la interpretación se debe implementar para cada caso.

Dentro del mercado se pueden encontrar los siguientes productos relacionados con la lectura y escritura en el bus CAN.

3.2.1. KVaser:

El producto KVaser, permite acceder al bus CAN a través del puerto OBD II del vehículo.

Es capaz de leer todos los mensajes de bus CAN al que se conecta y, dependiendo del modelo concreto, los envía al ordenador mediante un cable, Wifi, o lo almacena en una tarjeta SD.

Ilustración 10 - KVaser

Está preparado para conectarlo directamente al vehículo y al ordenador. Incluye un software para controlar sus funciones.

Las principales desventajas de este producto son:

Un alto coste, que supera los 400€ para el modelo más básico, y la necesidad de un ordenador con el que manejarlo e interpretar los datos.

3.2.2. Interfaz bluetooth OBDII

Se conecta al puerto OBD II del vehículo y transmite lo que lee por bluetooth.

Es necesario tener un ordenador o una *tablet* con un programa específico con el que comunicarse por bluetooth.

Esta interfaz transmite por bluetooth algunos de los datos del bus CAN al dispositivo que está conectado. Este traduce los datos recibidos y los muestra por pantalla da forma gráfica.

Ilustración 11 - Interfaz bluetooth

Tienen un coste muy bajo (rondan los 25€) y un reducido tamaño.

Sin embargo, están centrados en leer mensajes de diagnosis o algunos parámetros como revoluciones por minuto o temperatura del motor, y no son capaces de leer todos los mensajes que se transmiten por el bus CAN de alta velocidad.

3.2.3. Sistema de diagnosis multimarca

Sirve para realizar diagnósticos en los vehículos. Se conecta al puerto OBDII del vehículo y es capaz de determinar el estado del coche y si tiene algún problema, indica cual es.

Se utilizan sobre todo en talleres, para diagnosticar los vehículos más modernos y cambiarles algunos parámetros internos como los kilómetros que quedan hasta la siguiente revisión o incluso para actualizar el propio software interno del vehículo.

Existen numerosos modelos en el mercado. Lo habitual es que cada dispositivo sea capaz de leer los datos de un fabricante determinado.

Existen también modelos con una gran base de datos que almacena información para diagnosticar varias marcas de vehículos.

Ilustración 12 - Dispositivo de diagnosis

La principal desventaja en este tipo de terminales tienen un precio bastante elevado (alrededor de 1.500 €), además, encontramos que sólo se pueden utilizar para tareas de diagnosis y no para leer o escribir los mensajes que se desee.

4. Hardware del sistema

Para desarrollar este proyecto, son necesarios al menos dos componentes de hardware: una unidad que permita leer y escribir en el bus CAN, y otra unidad que la controle y sirva de interfaz entre la red y el usuario.

4.1. CANdiy-shield

Es el módulo que se encarga de leer y escribir en el CAN bus, y se comunica con la Raspberry Pi mediante SPI.

Consta principalmente de 2 microchips: el MCP2515 y el MCP2562, y de la electrónica necesaria para que funcionen: resistencias, diodos, condensadores y un reloj de cuarzo de 16MHz.

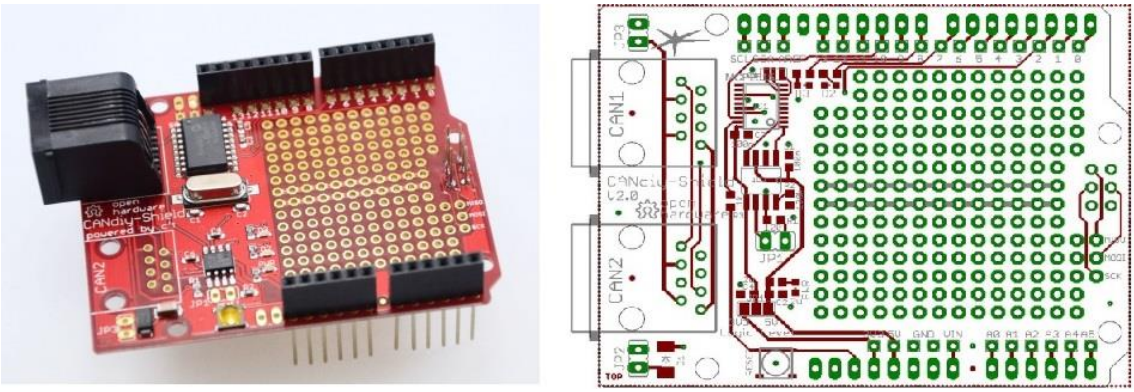


Ilustración 13 - Placa CANdiy-shield

Su elección está motivada por su bajo precio (unos 13€), y porque cumple con las especificaciónes requeridas: puede leer en una velocidad de hasta 1Mbps, y se puede controlar con un protocolo estándar (SPI), que cualquier computador pueda soportar.

4.1.1. Acondicionamiento a 3.3V

Dado que se va a conectar con la Raspberry Pi y esta funciona con un nivel de señal de 3.3V, es necesario hacer una modificación para que tambien lo haga el microchip.

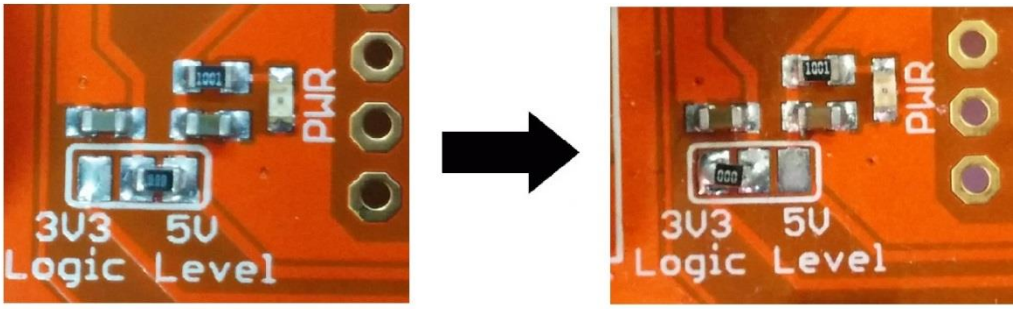


Ilustración 14 - Acondicionamiento a 3.3V

Dicha modificación consiste en cambiar de lugar una resistencia de la placa CAN-diy como se indica en la Ilustración 15.

Ahora el microchip MCP2515 está alimentado a 3.3V, y se podrá comunicar con la Raspberry Pi sin problemas.

4.1.2. Circuito de RESET

En la placa, el pin RESET del microchip MCP2515 no viene conectado a ningún sitio, ya que se puede implementar un circuito (descrito en la hoja de características) para realizar el reset por hardware al encender la placa.

Como dicho reset se va a hacer por software, el pin reset se va a conectar directamente a 3.3v (el pin reset es activo a nivel bajo), de forma que nunca se activa el reset con ese pin.

4.1.3. Microchips:

A continuación se va a explicar el funcionamiento de los dos microchips que componen la placa CANdiy-shield.

4.1.3.1. MCP2562

Es una interfaz de alta velocidad y tolerante a fallos entre el controlador (microchip MCP2515) y la capa física del bus CAN.

Cada nodo en un sistema CAN, debe tener un dispositivo que convierta las señales generadas por el controlador CAN a señales capaces de transmitirse sobre el cableado del bus. Además, este dispositivo sirve como buffer entre el controlador y los picos de corriente que se pueden producir en el bus CAN generados por otros elementos.

Los pines CANH y CANL se conectan a los cables correspondientes del bus CAN, y en los terminales TXD y RXD de este microchip se obtiene una señal compatible con el microchip MCP2515.

4.1.3.2. MCP2515

Es microchip principal, encargado de enviar y recibir los mensajes mediante el protocolo CAN.

Consta de 2 buffers donde se almacenan los mensajes recibidos, y otros 3 donde se almacenan los que se van a enviar. Para controlarlo, es necesario conectarlo a un dispositivo maestro que soporte la comunicación por SPI.

4.1.4. Comunicación SPI:

Para poder leer y escibir en la memoria del microchip, se debe usar el protocolo de comunicación SPI. Dicho protocolo permite la comunicación entre un módulo maestro y uno o varios esclavos. Consta de:

- Dos cables de comunicación: uno de recepción y otro de envío de datos (SI y SO).
- Uno de sincronización mediante pulsos de reloj, que son enviados por el módulo maestro (SCK).
- Otro de selección. Es enviado por el módulo maestro hacia el módulo con el que se quiere comunicar (CS).

Para que se produzca la comunicación, el módulo maestro debe activar la señal CS (activa a nivel bajo) y enviar pulsos de reloj. En cada pulso, tanto el módulo maestro como el esclavo pueden enviar y recibir datos al mismo tiempo, ya que se usan dos cables de transmisión diferentes.

En el caso concreto del MCP2515, la manera de comunicarse es enviando instrucciones de 8 Bits.

Si la instrucción es de lectura, habrá que enviar tambien la dirección de memoria en la que se quiere empezar a leer. Una vez enviada la dirección, en cada pulso de reloj se irá recibiendo la información de la memoria del microchip.

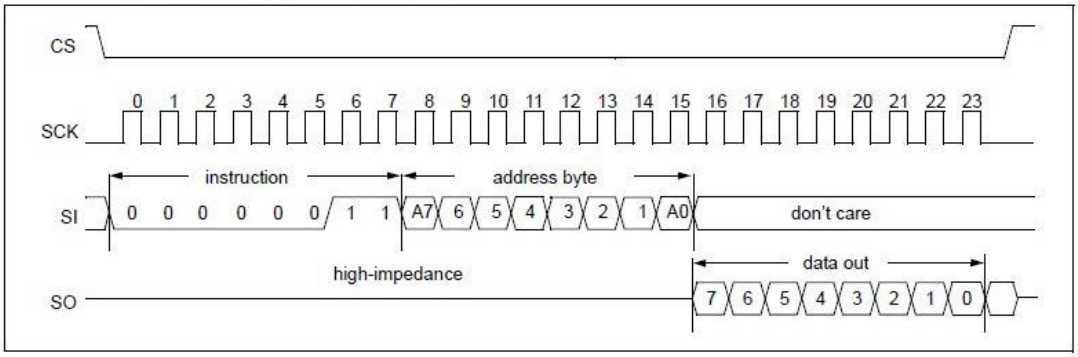


Ilustración 15 - Instrucción de lectura por SPI

Si la instrucción es de escritura, hay que indicar la dirección en la que se desa esacribir, así como los datos que se quieren enviar.

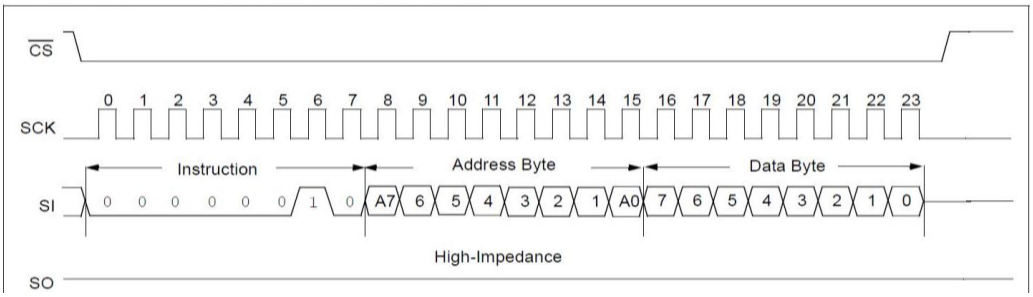


Ilustración 16 - Instrucción de escritura por SPI

Si una vez escritos o leidos los 8 bits de la memoria accedida, se siguen enviando pulsos de reloj, se accederá a la siguiente dirección de memoria. De este modo, para leer o escribir en memorias que están seguidas, no es necesario especificar la dirección de cada byte, basta con indicar la primera y mandar todos los bytes necesarios seguidos.

Para enviar o recibir mensages en el bus CAN, hay que seguir los siguientes procesos.

4.1.4.1. Recepción de mensajes:

Cuando el microchip recibe un mensaje, este se almacena en un buffer que esté libre teniendo mayor prioridad el primero.

Una vez almacenado en el buffer, se activa un flag que indica que se ha recibido un mensaje. Cada buffer tiene su propio flag asociado, por lo que se puede saber donde se ha almacenado el mensaje.

El proceso de lectura consiste en leer todos los bits del buffer donde se encuentran el "arbitration field", la longitud en Byites del mensaje, el tipo de mensaje (estándar, extendido o reemoto) y los datos del mensaje.

4.1.4.2. Envío de mensajes:

Para enviar un mensaje hay que hacer el proceso contrario al de lectura.

En primer lugar, hay que escribir el mensaje, asi como el resto de datos ("arbitration field", la longitud en Byites del mensaje, el tipo de mensaje) en el buffer desde el que se desea enviar.

Acto seguido, hay que activar un flag que le indica al microchip que debe enviar cuando pueda el mensaje. Cuando el mensaje se ha enviado, dicho flag se resetea.

4.1.4.3. Reseteo del microchip.

Cada vez que se enciende el microchip, es recomendable hacer un reset. Este reset se puede hacer de dos maneras: poniendo un circuito que mantenga el reset activo (a nivel bajo) hasta que se encienda el microchip, o de forma manual enviando una instrucción mediante SPI.

Se ha usado esta última opción ya que la placa CAN-diy no disponía de este circuito, de modo que, como se he comentado antes, el pin RESET del MCP2515 va conectado directamente a 3.3V.

Cada vez que se encienda el microchip habrá que enviar la instrucción de reset.

4.2. Conexiones:

Se utilizan dos tipos de cables para conectar los distintos elementos.

Para conectarse con la red de CAN bus, la placa CAN-diy shield utiliza un cable que tiene unos conectores específicos, y para conectar la placa CAN-diy shield con la computadora central se utiliza un bus de 26 pines con el que se conectan las salidas y entradas punto a punto.

4.2.1. Cable CAN bus - CAN-diy shield

Este cable conecta el bus CAN con la CAN-diy shield.

Para ello, un lado debe tener la forma de RJ-45 y el otro la del puerto OBDII. Además los pines de cada lado del cable deben estar conectados de una manera específica para que funcione.

Como no se ha encontrado ningún cable de este tipo, se ha optado por fabricarlo empalmando dos mitades, una de cada tipo y uniendo los cables de la forma requerida.

4.2.1.1. Conexionado RJ-45

Este extremo es el que va conectado a uno de los puertos de la placa CAN-diy shield.

El conector RJ-45 es muy utilizado para las conexiones de red de las computadoras. Su uso más común es en el cable de red de Ethernet.

De los 8 pines que tiene el RJ-45, solo 5 están conectados a la placa CAN-diy shield:

- 2 cables de tierra.
- Un cable de +12Voltios.
- Un cable para el CANH.
- Uncable para el CANL.

Aunque solo se van a utilazar los cables CANH y CANL, se han identificado el resto por si es necesario tener +12 Voltios de alimentación en el sistema en futuras aplicaciones.

La correspondencia del cable del RJ-45 con la señal que lleva es la se muestra en la Ilustración 17.

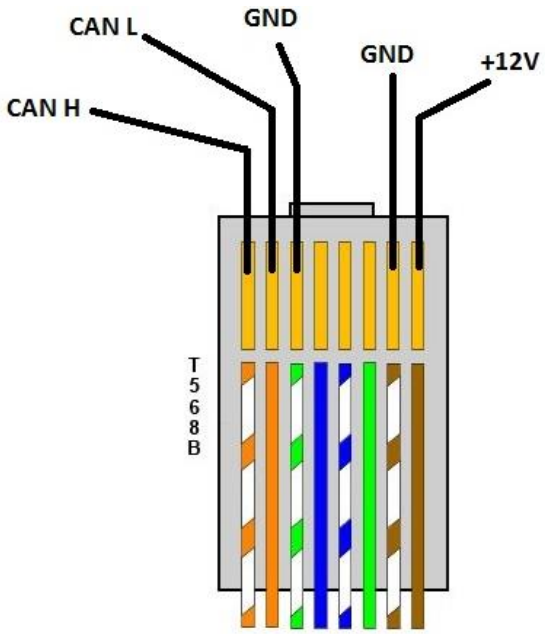


Ilustración 17 - Pines del cable RJ-45

4.2.1.2. Conexionado OBDII

El conector OBDII es una interfaz de hardware estandarizada pensada para conectar un ordenador de diagnosis y comprobar el estado del coche recogiendo información de los distintos protocolos de comunicación, a los que se puede acceder con este conector.

Dispone de 16 pines, 9 de los cuales están asociados a las mismas señales independientemente del vehículo. Los 7 pines restantes tienen funciones específicas que varían de un fabricante a otro.

Los 9 pines comunes tienen la sigunte información:

- SAE J1850 positivo.
- SAE J1850 negativo.
- Tierra del chasis.
- Tierra de la señal.
- CANH.
- CANL.
- K-Line.
- L-Line.
- Volataje de la batería.

De estas, la únicas que nos interesan son el CANH y el CANL.

Dado que la señal del bus CAN es diferencial, no es necesaria la referencia de tierra o del voltaje de la batería, aunque por futuras aplicaciones se han identificado en el conector OBDII.

Los pines de cada señal están donde se indica en Ilustracion 18.

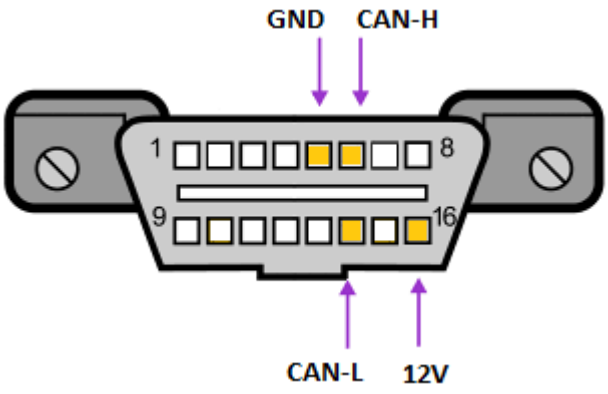


Ilustración 18 - Pines del conector OBDII

4.2.2. Conexiones CAN-diy shield con Raspberry Pi

Para las conexiones con la Raspberry Pi, se utilizan los siguentes pines de la placa CANdiy-Shield:

- 3V3: Corresponde a la alimentación de 3,3 Volios del MCP2515.
- 5V: Corresponde a la alimentación de 5 Voltios del MCP2562.
- GND: Es la tierra común de ambos microchips.
- CS (Chip select): corresponde al pin numero 12 del CAN-diy shield.
- MISO: Es la entrada de datos para el módulo mestro del SPI (Raspberry pi).
- MOSI: Es la salida de datos para el módulo mestro del SPI.
- SCK: Es la señal de reloj que sincroniza ambos módulos.

Todas estas conexiones se realizan con un bus de 26 pines que por un lado va conectado a la Raspberry Pi en el puerto GPIO, y por el otro va soldado a la placa CANdiy-shield pin a pin.

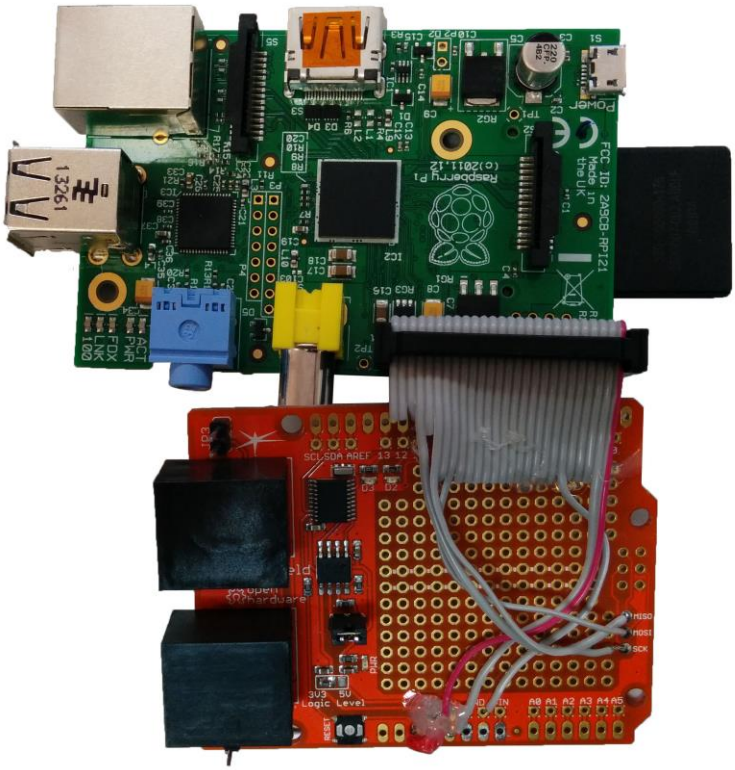


Ilustración 19 - Conexión entre RaspberryPi y CANdiy-shield

4.3. Computadora

Tras la selección del hardware que se encargará de enviar y recibir los mensajes que se transmitan por el bus CAN, el siguiente paso es elegir el hardware que se encargue de comunicarse con la placa anterior y mostrar una interfaz gráfica que permita al usuario realizar estas tareas de manera sencilla.

Como uno de los objetivos es realizar un dispositivo de bajo coste, pero a la vez suficientemente potente como para leer en el bus CAN de alta velocidad, es decir, lectura de velocidades de hasta 1Mbps, se ha optado por utilizar un computador embebido, que ofrecen con un tamaño y un precio muy reducido, una gran potencia.

Otra de las ventajas de los ordenadores embebidos es el puerto GPIO.

El puerto GPIO (General Purpose Input/Output o Entradas/Salidas de Propósito Peneral), son pines que tienen un comportamiento indefinido, el cual se puede programar por el usuario. Eso implica que estos pines pueden funcionar como entradas o salidas digitales, como salidas de modulación PWM, o como los pines necesarios para comunicarse mediante diferentes protocolos (SPI o I2C entre otros).

A la hora de elegir el modelo concreto, se han probado 2 opciones con diferentes precios y prestaciones:

4.3.1. Odroid X2

El Odroid X2, primera opción en este proyecto debido a su potencia, posee un procesador QuadCore de 1,2GHz, por lo que supera con creces la potencia necesaria para comunicarse con la placa Candiy-shield en tiempo real y además mostrar una interfaz gráfica.

Sin embargo, uno de los principales problemas de este computador está en sus puertos GPIO. Estos trabajan a un nivel de 1,8Voltios, mientras que los puertos de la placa Candiy-shield pueden funcionar a 3,3 o 5Voltios.

Para solventar este problema, se utilizaron dos tipos de conversores de nivel que se explican a continuación:

Ilustración 20 - Odroid X2

4.3.1.1. *Conversor bidireccional con transistores BSS138*

Su funcionamiento es muy simple, tiene dos entradas donde se fijan los dos niveles de tensión a los que se quiere convertir, y ocho pines para convertir las tensiones, cuatro para cada nivel.

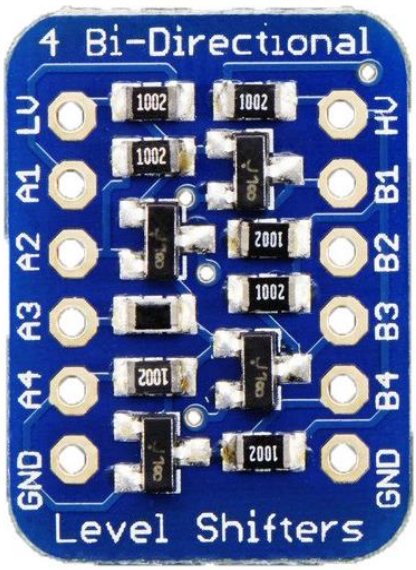


Ilustración 21 - Conversor de tensión BSS138

Además el propio microchip detecta si un pin se está utilizando como entrada o como salida.

Sin embargo, la frecuencia máxima a la que se necesita que funcione la comunicación entre la computadora y la placa Candiy-shield es de 20MHz, mientras que este microchip sólo permite una velocidad máxima de 200KHz.

Esto hace que no se pueda utilizar esta placa.

4.3.1.2. Conversor bidireccional con el circuito TXB0108:

El funcionamiento de este microchip es similar al anterior, sin embargo tiene 16 pines, 8 para cada nivel de tensión, y este si acepta velocidades de 20MHz.

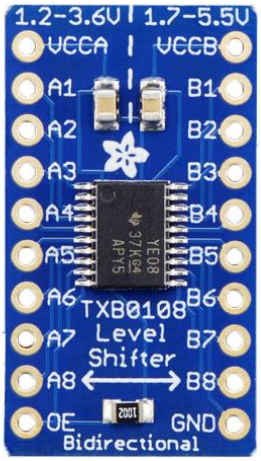


Ilustración 22 - Conversión de tensión TXB0108

Según la hoja de característica, este microchip soporta altas frecuencias. Sin embargo, produce un nivel de ruido muy alto en el lado de la placa Candiy-shield y ésta no responde siempre correctamente.

Por estos problemas entre otros, se buscó otra computadora alternativa con unas especificaciones parecidas.

4.3.2. Raspberry Pi

Se trata de un ordenador pequeño instalado en una única placa desarrollado por la fundación Raspberry Pi.

Su propósito inicial fue fomentar el aprendizaje de ciencias de la computación en colegios e institutos, pero debido a su bajo coste y altas prestaciones hoy en dia se utiliza en multitud de proyectos que necesitan una unidad de procesamiento de datos de tamaño reducido y barata.

Cuenta con un chip de la empresa Broadcom que incluye un procesador, una GPU y memoria RAM de diferentes características dependiendo del modelo en concreto de RPi.

Tiene un grupo de 28 a 50 GPIO (dependiendo del modelo concreto), que funcionan a un nivel de 3,3Voltios, por lo que se solventa el problema de la conversión de niveles.

Además de los puertos GPIO, la Raspberry Pi tiene puertos usb para conectar teclado, ratón o lo que se necesite, un puerto HDMI para conectar una pantalla y un puerto ethernet con el que se puede conectar a internet.

Existen distintos modelos con diferentes especificaciones. A continuación se muestran los dos más populares.

	Raspberry pi B	Raspberry pi 2
Chip	BCM2835	BCM2836
CPU	700 MHz	4nucleos 900MHz
RAM	512 Mb	1 Gb
Número de GPIO	26	50

Tabla 2 - Comparativa RPi B y RPi 2

Además de estos modelos, existen otros (modelo A y modelo B+) que tienen las mismas características que el modelo B y sólo varía el número de puertos USB y la capacidad de la memoria RAM.

Como se puede ver la Raspberry Pi 2 es más potente ya que es una nueva generación de este dispositivo y se ha sustituido el chip central BCM2835 por otro más potente BCM2836.

Aunque las especificaciones del modelo B son suficientes para el dispositivo que se desea desarrollar, el precio de ambas es el mismo, ya que la Raspberry pi 2 es una actualización del modelo anterior y se ha mantenido el mismo precio. Por lo tanto, se escogerá la Raspberry Pi 2 para el sistema.

El funcionamiento de las Raspberry Pi es muy parecido al de un ordenador convencional, se puede instalar un sistema operativo como Raspbian o Pidora (versiones de Debian y Fedora para la RPi) o incluso Windows 10 en la nueva Raspberry Pi 2. Como disponen de puertos USB, HDMI y Ethernet, se pueden conectar teclado ratón una pantalla y un cable de internet para utilizarlo como un PC cualquiera.

4.4. Pantalla táctil

Para que el dispositivo sea portable, se pensó que era necesario incorporar una pantalla, y para evitar tener que conectar un ratón y un teclado, dicha pantalla tendrá que ser táctil.

La pantalla debe ser de entre 6 y 8 pulgadas de tamaño, ya que tiene que ser suficientemente grande como para poder visualizar la interfaz y manejarla de forma correcta, y no puede ser muy grande ya que el dispositivo debe ser portable.

Ilustración 23 - Pantalla táctil para RPi

De entre varias alternativas, se ha elegido una pantalla TFT tactil de 7 pulgadas compatible con la Raspberry Pi. Dicha pantalla era la más económica (50,22€) que cumplía las especificaciones que se han explicado anteriormente.

Esta pantalla viene con un módulo que permite una entrada de imagen HDMI, DVI o VGA.

Además incluye un periférico que se conecta a la Raspberry Pi por usb que permite controlar el ratón de forma táctil en la pantalla.

4.5. Fuente de alimentación

Para alimentar el sistema, es necesario proporcionar dos tensiones diferentes, una para alimentar la Raspberry Pi y otra para la pantalla táctil:

- 5 Voltios para la RPi.
- 12 Voltios para la pantalla táctil.

La placa CANdiy-shield se alimenta con dos tensiones, 5 y 3.3 Voltios, suministrados por la Raspberry Pi, por lo tanto, no es necesario que la fuente de alimentación proporcione esas tensiones.

Además, a la hora de elegir la fuente de alimentación, hay que tener en cuenta la corriente mínima que necesita cada elemento para funcionar de forma correcta:

- 450 miliamperios para la pantalla táctil.
- 900 miliamperios para la RPi junto con la placa CANdiy-shield.

Según el uso que se le vaya a dar a SIMBA, la elección de la fuente de alimentación es diferente.

Si se va a usar para analizar piezas de un vehículo, basta con usar un transformador convencional, que se conecte a la red doméstica de 230V y proporcione las tensiones deseadas (5 y 12 Voltios).

Si se va a utilizar dentro del vehículo para tomar datos mientras este está circulando, es más conveniente utilizar una fuente que se pueda conectar a la toma de mechero del coche o aprovechar los pines del puerto OBDII de +12V y GND.

4.6. Cubierta

Como se ha indicado en los objetivos, el dispositivo debe ser un sistema embebido, que soporte conectarlo y desconectarlo sin que se deteriore, ya que la única sujeción entre la placa CANdiy-shield y la Raspberry Pi son unos cables soldados.

El prototipado se ha llevado a cabo, fijando los elementos en una cubierta de plástico, para establecer una primera versión, la cual, ya se ha utilizado en numerosos experimentos del laboratorio LSI para la adquisición de datos.

Dicha cubierta es lo suficientemente grande para que quepan todos los elementos, y es de un material fácil de modificar para poder realizar rectificaciones y colocar los conectores en la superficie de la cubierta.

Ilustración 24 - Cubierta

5. Software del sistema

En este apartado se describe el funcionamiento del software así como su estructura.

5.1. Desarrollo en Raspbian

Para desarrollar el proyecto en la Raspberry Pi, se ha utilizado el sistema operativo Raspbian.

Raspbian (Raspberry + Debian) es un sistema operativo basado en Debian, pero optimizado para el procesador de Raspberry Pi.

Para programarlo, funciona como un PC convencional con cualquier distribución de Linux, es decir, basta con escribir un programa en el lenguaje deseado y compilarlo. La única diferencia reside en el manejo del puerto GPIO, que se controlan modificando los registros del microprocesador, normalmente a través de una serie de drivers y librerías.

Para utilizar la comunicación SPI en el puerto GPIO, es necesario activarla mediante las opciones del menú raspi-config, y además hay que instalar unos drivers que permitan utilizar la comunicación SPI de forma más fácil.

5.2. Instalación y funcionamiento de los drivers

Para instalar los drivers, es necesario descargarlos de la página airspayce.com.

Los drivers utilizados se llaman bcm2835, al igual que la CPU de la Raspberry Pi.

Dichos drivers contienen una librería que da soporte, al control del puerto GPIO en general, como comunicación SPI e I2C, distintas opciones para controlar las entradas y salidas analógicas y salida PWM entre otros.

Las funciones específicas para SPI de las librerías bcm2835 acceden a los registros del microprocesador y modifican los registros correspondientes para enviar o recibir datos a través de dicho protocolo.

Existen una serie de funciones que sirven para configurar parámetros del protocolo como la velocidad, el modo de envío o el pin de selección que se va a usar, y otras que se encargan de enviar/recibir los datos que se transmiten.

Para controlar la placa CANdiy-shield, he creado una librería de funciones que permitan tanto configurarla como leer y enviar mensajes en el bus CAN de forma más fácil.

5.3. Librería de funciones.

Para controlar la placa CANdiy-shield, existen tres instrucciones básicas: Leer en una dirección de memoria, escribir en una dirección de memoria y modificar bits.

A partir de estas instrucciones básicas he creado funciones para leer los buffers de entrada, escribir en los de salida y dar instrucciones a la placa CANdiy-shield de configuración y ordenes de lectura y escritura.

5.3.1. Función de lectura

Para leer los mensajes recibidos en el buffer de la placa CANdiy-shield, se ha creado una función que lee todos los bytes del buffer y los separa en los distintos campos que tiene un mensaje CAN.

El proceso que se sigue para leer un mensaje es el siguiente:

- En primer lugar lee los parámetros de configuración como la velocidad del bus CAN o los datos del filtro.
- Después comprueba dos bit que indican si en un buffer se ha recibido algún mensaje y en cual.
- Si cumple los criterios del filtro, o no se ha activado el filtro, se escribe el mensaje.

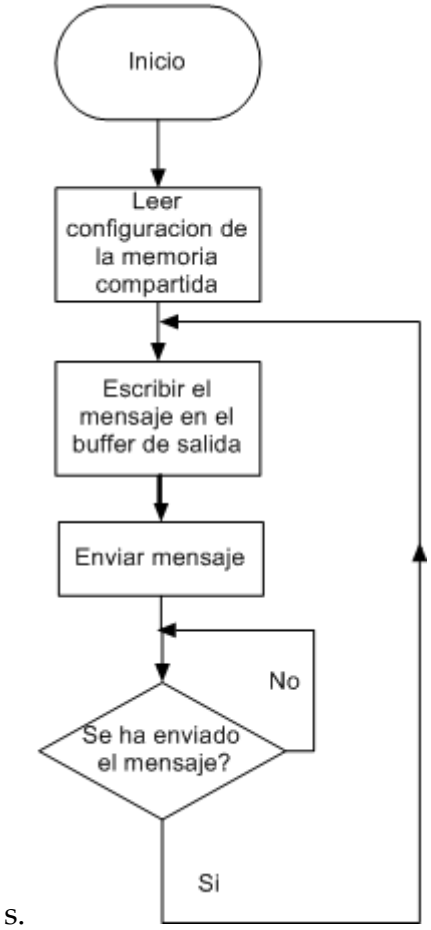
Ilustración 25 - Diagrama de flujo de lectura

5.3.2. Función de escritura

Para escribir mensajes en el bus CAN se ha creado otra función que almacena los datos de los campos del mensaje CAN en los Bytes del buffer de salida.

Para enviarlos, se sigue el siguiente proceso:

- Se lee la configuración, al igual que para leer mensajes y se escriben los datos de los campos del mensaje CAN un buffer de salida.
- Para dar la orden de enviar el mensaje, hay que activar un bit asociado al buffer que se quiere enviar.
- Cuando se envía, este bit se resetea, indicando que podemos continuar enviando mensaje.



s.

Ilustración 26- Diagrama de flujo de escritura

5.3.3. Función de configuración

Esta función se ejecuta siempre al iniciar el sistema, después de hacer el reset del MCP2515.

Se incluyen parámetros de la configuración del can bus como la velocidad, la función de algunas entradas del microchip o el uso de filtros internos y máscaras a la hora de recibir mensajes.

Para modificar estos parámetros hay que modificar algunos registros de la memoria del microchip.

Para indicar la velocidad del bus CAN, hay que indicar la duración de todas las partes del tiempo nominal de Bit (2.4.), con respecto a la velocidad del oscilador de cuarzo instalado en la placa (en este caso de 16MHz).

5.4. Programa modular

Para facilitar futuras modificaciones, se ha dividido el programa en distintos módulos con unas funciones específicas cada uno.

Además, para que dichos módulos se comuniquen entre sí, se ha definido una zona de memoria compartida, donde intercambia la información necesaria para su correcto funcionamiento.

Cada módulo es un programa independiente. Cuando se quiere alguna de las funciones implementadas (leer, escribir, o demostración), se lanza el módulo asociado desde la interfaz gráfica de tal forma que ambos están funcionando a la vez en paralelo. Si se está ejecutando un módulo, la interfaz no permite que se lance otro hasta que el primero se pare, ya que podrían entrar en conflicto entre sí. Para parar un proceso, la interfaz le envía una señal y éste ejecuta una serie de acciones (como cerrar los archivos txt o borrar la memoria) y termina su ejecución.

El sistema consta de los siguientes módulos:

5.4.1. Módulo de la interfaz gráfica

Es el módulo principal se encarga principalmente de dos tareas:

Facilitar al usuario el control del bus CAN, ofreciendo al usuario una interfaz con botones y cuadros de texto con los que pueden controlar todas las opciones que ofrece el sistema.

La segunda tarea de la interfaz gráfica es coordinar la ejecución de los demás módulos, así como su finalización.

Este módulo crea una zona de memoria compartida que sirve para que la interfaz y el resto de módulos compartan información entre sí, ya que todos tienen acceso a ella.

Todas las opciones de configuración que modifica el usuario en la interfaz gráfica, se guardan inmediatamente en la memoria compartida correspondiente, de tal forma que cuando se ejecuta un proceso que accede al

CAN bus, éste lee de la memoria compartida los parámetros de su configuración.

Para desarrollar la interfaz gráfica, se ha utilizado Qt Creator. Este programa permite desarrollar una interfaz utilizando C++, e incluye una gran variedad de clases para implementar todos los elementos necesarios de la interfaz así como una pantalla donde se visualiza el resultado final.

5.4.2. Módulo de lectura:

El módulo de lectura se encarga de leer todos los mensajes del CAN bus y guardarlos en un archivo de texto. Se puede elegir la ubicación y el nombre del nuevo archivo en la interfaz gráfica con el botón "Browse".

Cuando se pulsa "Read CAN" se lanza este módulo que se ejecuta en segundo plano y se mantiene activo leyendo datos hasta que se pulsa "Stop".

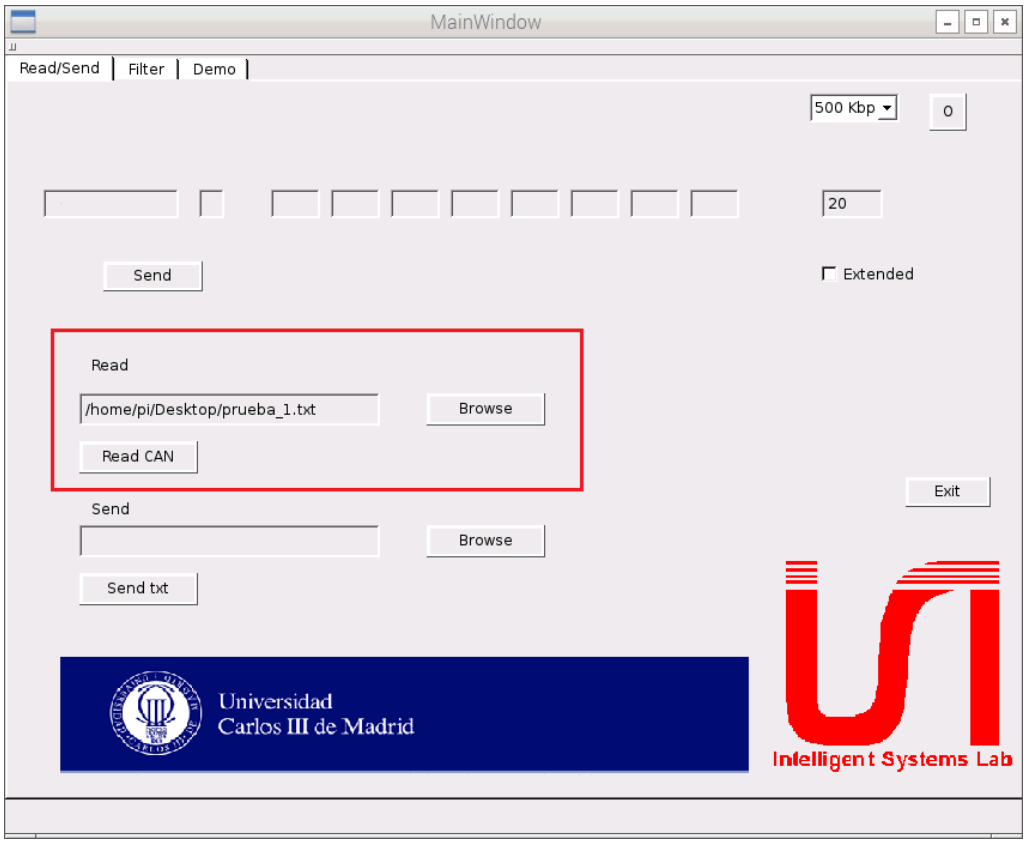


Ilustración 27 - Escribir en archivo de texto

Como en el bus CAN pueden haber muchos mensajes, existe una opción que filtra los mensajes para solo escribir los que tengan un "arbitration field" determinado. Esta opción se puede configurar en la pestaña Filtro de la interfaz gráfica.

Si se selecciona el campo "Filter data", solo se escribirán en el archivo de texto los mensajes que tengan un valor en la cabecera igual a alguno de los que están escritos en los 7 campos de esta pestaña.

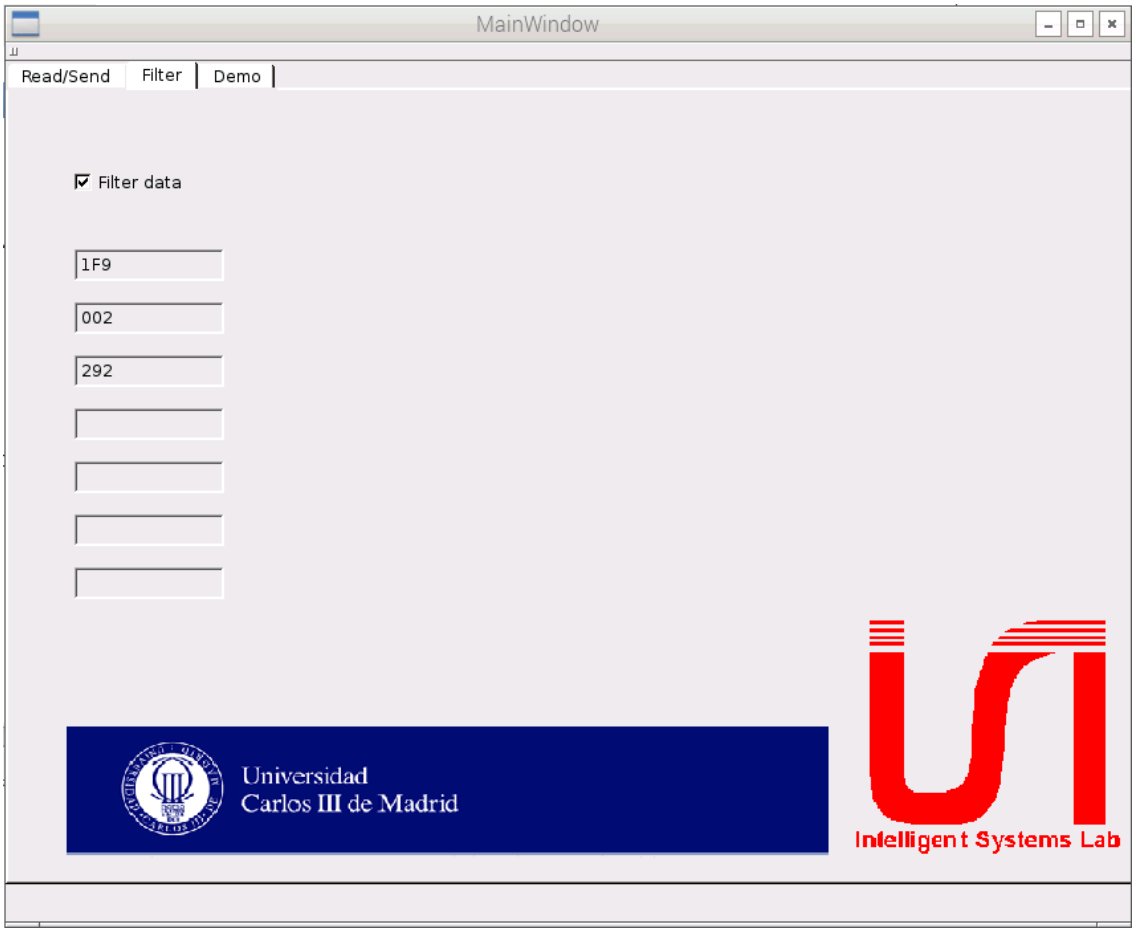


Ilustración 28 - Filtrar mensajes

5.4.3. Módulo de escritura de un mensaje:

Este módulo de escritura permite enviar un mensaje determinado cada cierto intervalo de tiempo. Todos los parámetros del mensaje, así como la frecuencia de envío, se pueden indicar en la interfaz.

El primer campo corresponde a la cabecera del mensaje que se desea mandar, el segundo indica la longitud del mensaje y los ocho siguientes los datos del mensaje y el último sirve para establecer cada cuantos milisegundos se va a enviar el mensaje.

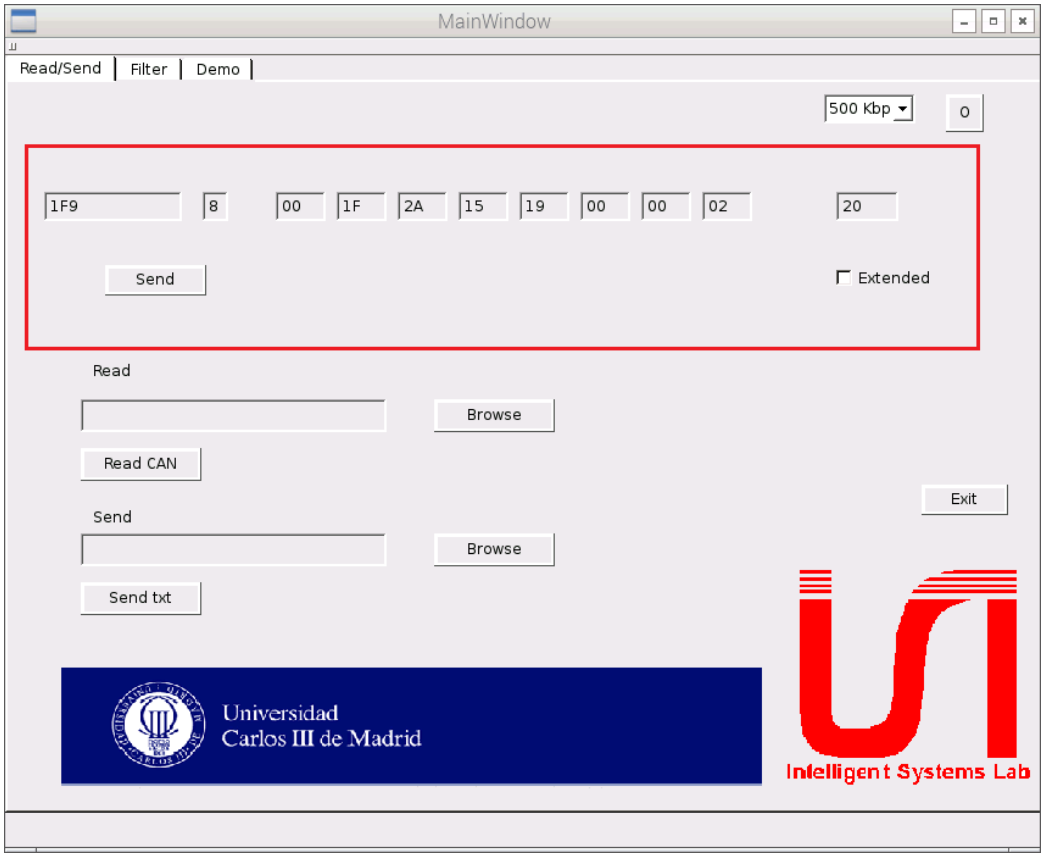


Ilustración 29 - Mandar mensaje único

Este módulo es de gran utilidad para comprobar la función que tiene un mensaje determinado en un vehículo.

5.4.4. Módulo de escritura múltiple

El módulo de escritura múltiple permite enviar una serie de mensajes almacenados en un archivo de texto. Es posible guardar una secuencia de datos con el módulo de lectura y luego reproducirlos con este módulo, de forma que se puede simular el bus CAN de un vehículo determinado.

Con el botón "Browse" se selecciona el archivo fuente que va a ser enviado y con "Send txt" se empiezan a enviar los mensajes este archivo hasta que se pulsa "Stop".

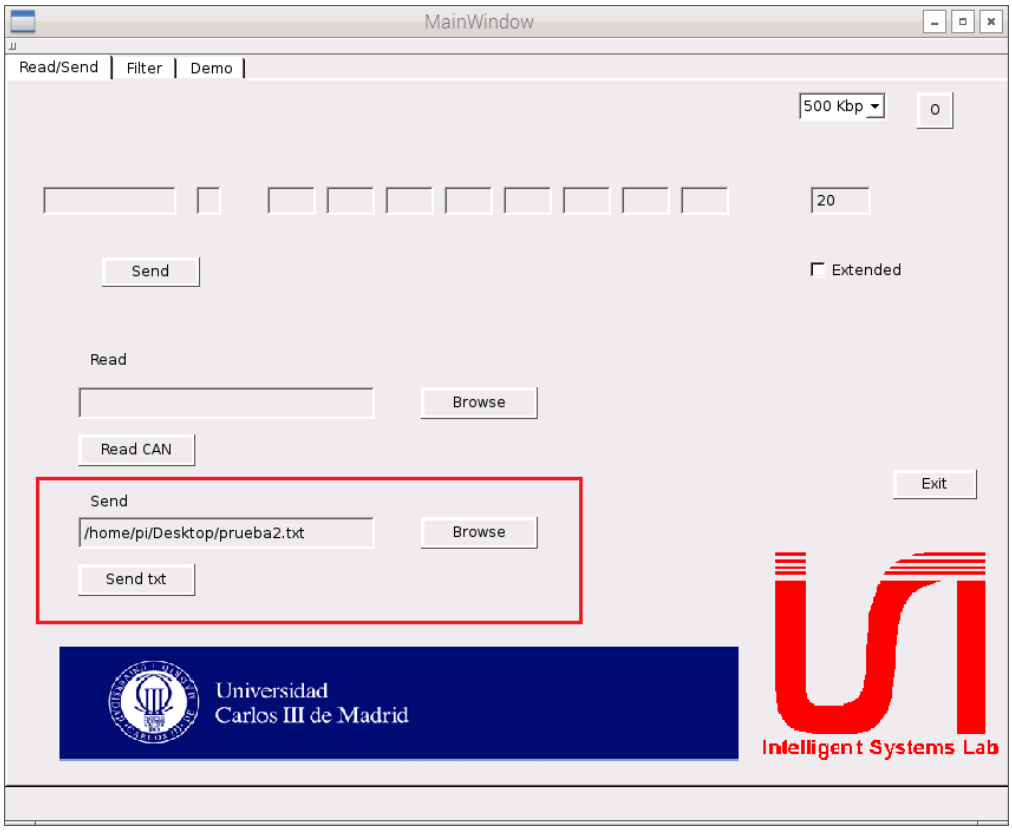


Ilustración 30 - Mandar varios mensajes

5.4.5. Módulo demostración

Este último módulo sirve para hacer una demostración visual de los datos que se pueden leer del vehículo. Cuando lee un mensaje del ángulo del volante o uno de las revoluciones por minuto, lo guarda en la memoria compartida. Por otra parte, la interfaz gráfica lee estos valores cada 200 milisegundos y los

muestra en pantalla de modo gráfico girando un volante los mismos grados que el real y moviendo una barra que representa las revoluciones.

Ilustración 31 - Módulo demo

5.5. Programa para enviar los datos por Red

Todos los datos que se leen por el bus CAN se muestran o se almacenan a través de la Raspberry Pi, de manera que si es necesario abrirlos en otro ordenador, para postprocesarlos por ejemplo, hay que copiar los archivos leídos en una memoria extraíble. De esta forma no es posible utilizar los datos leídos en tiempo real por otra computadora.

Para solucionar esto, se ha desarrollado un programa que envía los datos leídos por el bus CAN mediante un protocolo TCP/IP a otro programa que actúa como servidor y está en un ordenador distinto.

Para que funcione correctamente, en primer lugar hay que ejecutar el programa que actúa como servidor en el otro ordenador.

Para ello hay que abrir el ejecutable indicando el puerto que se va a utilizar. Por ejemplo, si se va a utilizar el puerto 3005, el comando para ejecutar el programa sería:

$ ./servidor 3005

De esta forma se inicializa el servidor y se queda esperando a que se conecte alguien a él.

Ahora, se puede ejecutar el programa de la Raspberry Pi que enviará los datos del bus CAN mediante el protocolo TCP/IP.

Para ejecutarlo, hay que indicar la dirección IP del servidor, y el puerto que se ha utilizado en el servidor. Por ejemplo, si la dirección IP es 163.117.155.55, el comando con el que habrá que ejecutarlo es:

$ sudo ./enviar_TCPIP 163.117.155.55 3005

El funcionamiento del programa de la Raspberry Pi es similiar al módulo que escribe los datos del bus CAN en un archivo de texto.

Nada más ejecutarse el programa, se conecta al servidor, que ya se está ejecutando. Cada vez que recibe un mensaje, en vez de escribirlo en un archivo de texto, lo almacena en una estructura de datos que envía mediante un socket al servidor.

Como para los datos se utiliza un protocolo TCP/IP, se pueden enviar tanto de forma inalámbrica (WIFI), como por cable (Ethernet), basta con configurar la red correctamente en la Raspberry Pi.

6. Resultados

El resultado final (SIMBA) es un dispositivo compacto con el que se pueden realizar distintas tareas de lectura y escritura en el bus can de cualquier vehículo que disponga de un conector OBDII.

Ilustración 32 - Dispositivo final

Cuenta con:

- Una entrada para la fuente de alimentación junto a un interruptor que enciende o apaga todo el sistema.
- Dos salidas USB por si es necesario conectar un teclado, un ratón, o cualquier otro dispositivo USB.
- Una salida HDMI para conectar una pantalla externa si es necesario.
- Un puerto Ethernet donde se conecta el cable OBDII – Ethernet para leer de bus CAN.

Se puede manejar con la interfaz gráfica a través de la pantalla táctil, de forma que para que el dispositivo funcione, tan solo hay que conectar el cable de alimentación.

Para probar el dispositivo diseñado, se han realizado distintas pruebas en vehículos reales y con piezas de coches.

6.1. Lectura de datos del IVVI

Se han recogido los datos transmitidos por el bus CAN del coche del LSI IVVI 2.0.

Se hizo una toma por ciudad y carretera de aproximadamente una hora, en la que el dispositivo guardó los datos en un archivo de texto como el de la Ilustración 33.

Arbitration field	Longitud	Datos								Fecha	Microsegundos
00000284	8	00	00	00	00	00	00	04	8a	27-03-2015	11:40:07 483309
00000244	7	fe	fe	00	1a	00	10	fe	00	27-03-2015	11:40:07 483535
00000354	8	00	00	00	00	20	00	00	00	27-03-2015	11:40:07 483783
00000181	8	18	fa	39	10	27	25	3d	02	27-03-2015	11:40:07 484842
00000351	8	1c	02	33	10	00	00	10	00	27-03-2015	11:40:07 485079
0000035d	8	80	cb	12	00	08	ff	12	00	27-03-2015	11:40:07 486762
00000300	1	00	00	00	00	00	00	00	00	27-03-2015	11:40:07 487105
000001f9	6	20	1e	18	fa	0a	ff	00	00	27-03-2015	11:40:07 488809
000002de	8	00	00	80	ff	ff	ff	07	8d	27-03-2015	11:40:07 490307
00000002	5	df	ff	00	07	63	00	00	00	27-03-2015	11:40:07 502111
00000161	5	39	27	65	00	10	00	00	00	27-03-2015	11:40:07 502562
000002a0	5	c8	7f	ff	7f	fd	00	00	00	27-03-2015	11:40:07 503194
00000161	5	39	27	65	00	10	00	00	00	27-03-2015	11:40:07 503465
00000244	7	fe	fe	00	1a	00	10	fe	00	27-03-2015	11:40:07 504052
000005c5	8	44	00	a9	ce	00	0c	00	7f	27-03-2015	11:40:07 504561
00000625	6	02	00	e2	1d	00	00	00	00	27-03-2015	11:40:07 506129
00000300	1	00	00	00	00	00	00	00	00	27-03-2015	11:40:07 507578
000001f9	6	20	1e	18	fc	0a	ff	00	00	27-03-2015	11:40:07 508813
000002de	8	00	00	80	ff	ff	ff	07	8d	27-03-2015	11:40:07 510206
00000002	5	df	ff	00	07	05	00	00	00	27-03-2015	11:40:07 510918
00000161	5	39	27	65	00	10	00	00	00	27-03-2015	11:40:07 511672
00000181	8	18	fc	39	10	27	25	3d	02	27-03-2015	11:40:07 514849
00000215	6	a3	30	3c	02	ff	ff	00	00	27-03-2015	11:40:07 518079
000001f9	6	20	1e	18	fc	0a	ff	00	00	27-03-2015	11:40:07 525922
00000181	8	18	fc	39	10	27	25	3d	02	27-03-2015	11:40:07 526875
00000551	7	57	48	64	80	ff	72	00	00	27-03-2015	11:40:07 527530
00000300	1	00	00	00	00	00	00	00	00	27-03-2015	11:40:07 528260

Ilustración 33 - Ejemplo de datos tomados

Una vez almacenados todos los mensajes del bus CAN, es necesario postprocesarlos para conocer el significado de los mismos.

Dado que los fabricantes de automóviles no proporcionan información sobre el significado de estos mensajes, es necesario analizarlos uno a uno para averiguar la información que contienen.

Por ejemplo, tras analizar los mensajes de algunas cabeceras, se descubre que en el "00000002", los dos primeros bytes indican el ángulo de giro del volante.

Durante esta prueba, el sistema demostró ser muy robusto, ya que almacenó datos durante más de una hora sin problemas.

Además, durante la prueba se desconectó el cable OBDII y al reconectarlo el sistema continuó guardando datos sin problemas, demostrando una gran tolerancia a posibles fallos.

6.2. Envío de mensajes al IVVI

También se ha probado a enviar mensajes al IVVI para controlar algunas funciones del vehículo.

Se ha logrado controlar las luces de posición, encenderlas y apagarlas desde el dispositivo, así como controlar los pilotos de información del cuadro del coche, como el que indica si está activo el intermitente.

Además, se ha tratado de enviar mensajes para controlar otras partes más importantes como el acelerador.

Para ello se han identificado los mensajes que mandan información del acelerador y se han guardado mientras se acelera el coche.

Después se han enviado, pero el coche no responde, y enciende un piloto de error en el cuadro de mandos.

Esto puede deberse a que el acelerador real del coche está enviando mensajes contradictorios a los que envía SIMBA y entran en conflicto.

6.3. Simulación de un entorno CAN

Otra de las pruebas que se han realizado, es la de simular un entorno CAN en una pieza de un vehículo para que se comporte como si estuviera conectada a este. La pieza en concreto que se ha utilizado es una pieza de dirección asistida, que cuando está conectada al vehículo y este está encendido, debe asistir al giro.

Para simular el entorno del vehículo, se ha conecta SIMBA al vehículo del que procede la pieza y se han almacenado los mensajes que envía el vehículo encendido por el bus CAN en un archivo de texto.

Después se ha conectado la pieza de dirección a SIMBA y se han enviado los mensajes almacenados en el archivo de texto anterior.

Se observa como cuando se están enviando los datos recogidos con el vehículo encendido, la pieza asiste al giro y cuesta mucho menos girar la barra de dirección que cuando se para la simulación.

7. Trabajos futuros

A continuación se explicarán algunas mejoras que se pueden incluir al dispositivo, así como otras funcionalidades que añadir al sistema.

SIMBA proporciona funciones de acceso al bus CAN genéricas (leer y escribir). Los trabajos que se enuncian a continuación son funcionalidades orientadas a una aplicación concreta que se pueden agregar al dispositivo.

Estas nuevas mejoras en las que trabajar son las siguientes:

7.1. Comunicación inteligente:

Puede ser necesario que los mensajes que hay que enviar dependan de los que recibe del vehículo, por lo tanto, una posible mejora para el futuro es implementar otro módulo que sea capaz de responder ante ciertos mensajes del vehículo con otros que dependan de los recibidos.

7.2. Ayuda para analizar los mensajes CAN

Como se ha explicado anteriormente, analizar los datos que se envían por el bus CAN del vehículo después de tomarlos es una tarea compleja.

Por lo tanto se podría desarrollar un módulo, que grafique en tiempo real por la pantalla del dispositivo, los datos de los diferentes mensajes. De esta forma, al variar algún parámetro del vehículo (por ejemplo pisar el freno) se puede observar qué gráfica está variando, según la variación del parámetro del vehículo y se puede identificar qué mensajes son los que informan de dicho parámetro.

7.3. Mejora de la cubierta

Otro punto en el que seguir trabajando es la cubierta. Para este proyecto, se ha realizado un prototipo metido en una cubierta mucho más grande de lo necesario.

Analizando el problema, es posible reorganizar todas las piezas y meterlas en una cubierta mucho más pequeña y por lo tanto más portable.

7.4. Enviar datos a ROS

Por último, otra funcionalidad que se puede implementar en el futuro es el envío de datos del vehículo al ordenador central del IVVI.

Por el bus CAN de este coche circulan una gran cantidad de datos que pueden ser de gran utilidad para fusionarlos con los que proporcionan los sensores incorporados al IVVI (cámaras, laser, IMU, etc). Por ejemplo, para el sistema que detecta cambios de carril a través de una cámara, puede ser útil la información de los intermitentes para detectar si los cambios son voluntarios; o el sistema de detección de peatones con cámara y laser puede necesitar la posición del volante para centrarse más en la zona hacia la que va a ir el vehículo.

Mediante el programa para enviar datos por red, se pueden enviar todos los datos leídos al ordenador central del IVVI llamado TESLA.

Tesla funciona con el sistema operativo ROS, por lo tanto, para que el sistema funcione correctamente, hay que modificar el programa que actúa como servidor para que filtre y publique los datos importantes del bus CAN en ROS.

La Raspberry Pi se puede conectar al router central del IVVI mediante un cable Ethernet, o mediante una llave WIFI USB.

8. Presupuesto

En este apartado se describe el coste total del proyecto.

Se ha dividido en tres secciones, una dedicada al coste del hardware del dispositivo en sí, y otras dos que informan el coste relacionado con desarrollar este dispositivo (de materiales y de personal).

8.1. Hardware del dispositivo

En este apartado se incluye el material que forma parte del dispositivo.

En la Tabla 3 se muestra el precio de cada elemento y el total.

Coste del material	
Elemento	**Precio (€)**
Raspberry Pi 2	35,03
CANdiy-shield	12,61
Cable RJ-45	2,25
Cable OBDII macho	5,76
Cable y conector de 26 Pines	2,95
Pantalla Táctil para Raspberry pi	50,22
Cable HDMI corto	5,85
Fuente de alimentación	14,90
Caja para prototipo	15,93
Coste total	145,50

Tabla 3 - Coste del hardware del dispositivo

8.2. Material necesario para realizar el proyecto.

Aquí se incluye la parte del material que es necesaria para desarrollar el proyecto, pero que sin embargo no forma parte de este y se puede utilizar para otro proyecto.

Para tener un valor más adecuado de este coste, se ha tenido en cuenta solo el coste de amortización de este equipo en el tiempo que ha durado el desarrollo del proyecto.

La amortización para sistemas informáticos es del 33%, para equipos electrónicos del 20% y para otros elementos del 10%.
Para calcular el precio e desamortización, he usado la siguiente fórmula.

$$Precio\ amortización = Precio * \frac{Porcentaje\ de\ amortización}{100} * \frac{5}{12}$$

Material necesario para el proyecto			
Elemento	**Precio**	**Porcentaje de amortización**	**Precio de desamortización (€)**
Pantalla LCD con HDMI	144,22	33%	19,83
Cable HDMI	7,99	20%	0,66
Teclado y ratón USB	20,90	20%	1,74
Ordenador de apoyo	768,23	33%	105,63
Soldador y estaño	30,55	10%	1,27
Coste total			129,13

Tabla 4 - Coste del material auxiliar

8.3. Coste de personal

En este apartado se incluye el coste que tendría un ingeniero trabajando las horas necesarias para desarrollar este proyecto.

Según la encuesta de salarios del 2014-2015 para ingenieros industriales, el salario medio para un ingeniero con menos de un año de experiencia trabajando en el sector de la investigación es de 18.119€ brutos al año.

La duración del desarrollo del proyecto ha sido de 5 meses.

Por lo tanto, el coste del ingeniero es:

Cálculo del salario		
Sueldo bruto Anual	**Duración**	**Coste (€)**
18.119€/año	5meses	7.549,58€

Tabla 5 - Cálculo del salario

8.4. Coste total del proyecto

La Tabla 6 muestra el coste de cada parte y el total del proyecto.

Presupuesto total	
Concepto	**Coste (€)**
Material del dispositivo	145,50
Material auxiliar	129,13
Salario del ingeniero	7.549,58
Coste total	7.824,21

Tabla 6 - Coste total del proyecto

9. Conclusiones

Se ha conseguido desarrollar un dispositivo que cumple con los objetivos marcados en el primer capítulo y funciona correctamente.

El sistema ha demostrado ser bastante estable, tolerando fallos como la desconexión accidental del bus.

Además se ha comprobado que el sistema funciona en tiempo real y tiene tiempo suficiente para leer o escribir todos los mensajes y no perder ninguno. De hecho, como muestra el módulo de demostración de la interfaz gráfica, es capaz de mostrar por pantalla estos datos sin ningún tipo de retardo.

Como se ha descrito en el presupuesto, el coste total del proyecto ha sido de 7.824,21€, sin embargo, la parte del hardware, sin incluir todo el coste del desarrollo, asciende a solo 145,50€, por lo que, comparado con las alternativas que hay en el mercado y en relación a las funcionalidades que ofrece, su precio es muy bajo.

Como se ha indicado anteriormente, el bus CAN es un protocolo muy utilizado actualmente en la gran mayoría de los vehículos del mercado por el que circula una gran cantidad de información.

Desarrollar una herramienta de bajo coste que permita acceder a este bus genera una gran cantidad de aplicaciones.

Estas aplicaciones pueden ser, entre otras:

- Aprovechar todos los datos de los sensores de los vehículos para combinarlos con otros sensores externos y tener más información del entorno.
- Controlar el vehículo mediante de forma remota. Con unas pocas modificaciones en SIMBA sería posible controlar los elementos básicos para manejar el coche (volante y acelerador).
- Crear una red bus CAN desde cero utilizando lo aprendido para diseñar nuevos nodos y utilizando SIMBA como computadora central.

Con este dispositivo se han creado las bases para seguir desarrollando cualquiera de estas posibilidades.

10. Bibliografía

[1] D. Martín, F. García, B. Musleh, D. Olmeda, G. Peláez, P. Marín, A. Ponz, C. Rodríguez, A. Al-Kaff, A. de la Escalera, J.M. Armingol, «ScienceDirect,» [En línea]. Available: http://www.sciencedirect.com/science/article/pii/S0957417414003947#gr2. [Último acceso: Junio 2015].

[2] Microchip, *MCP2515 - Datasheet.*

[3] A. Grzemba, «MOST The automotive Multimedia Network,» Franzis.

[4] R. B. GmbH, «CAN Specification,» 1991.

[5] «CAN-cia,» [En línea]. Available: http://www.can-cia.de/index.php?id=161. [Último acceso: Marzo 2015].

[6] «Kvaser,» [En línea]. Available: http://www.kvaser.com/products-services.

[7] «Auteltech,» [En línea]. Available: http://www.auteltech.com/Automotive%20Diagnostic&%20Analysi/341.jhtml.

[8] Microchip, *MCP2561/2 - Datasheet.*

[9] «airspayce,» [En línea]. Available: http://www.airspayce.com/mikem/bcm2835/.

[10] «eberspaecher-electronics,» [En línea]. Available: http://www.eberspaecher-electronics.com/en/company/flexray-introduction.html#c62713.

[11] «hanser-automotive,» [En línea]. Available: http://www.hanser-automotive.de/fileadmin/heftarchiv/FLEX_10_ONL_NXP-Y.pdf.

[12] «Coiig,» [En línea]. Available: http://coiig.com/COIIG/dmdocuments/Empleo%20LANBIDE/Salarios/encuesta_salarios_ingenieros_industriales_capv-navarra_2014-2015.pdf. [Último acceso: Junio 2015].

[13] M. Concepcion, Estrategias de Sistemas OBD II.

[14] «Vector Academy,» [En línea]. Available: http://elearning.vector.com/index.php?wbt_ls_kapitel_id=1330154&root=378422&seite=vl_flexray_introduction_en. [Último acceso: Junio 2015].

Anexo I

Configuración de la comunicación SPI en la Raspberry Pi

En este anexo se explicará cómo configurar la Raspberry Pi y qué modificaciones hay que hacer para que funcione correctamente la comunicación SPI.

En primer lugar hay que activar el protocolo SPI en la Raspberry Pi.

Para ello, escribir en la terminal:

$sudo raspi-config"

Esto abrirá un menú donde se pueden configurar distintos parámetros.

Dentro del menú de configuración, hay que dirigirse a la sección "Advanced options" y a continuación a "SPI".

Hay que activar la comunicación SPI e indicar que se active por defecto al arrancar.

Además, hay que activar desde este mismo menú la opción "device tree support".

Una vez hecho esto, se procede a instalar los drivers.

Para descargarlos hay que ingresar en la página web oficial (http://www.airspayce.com/), donde indica las distintas versiones que hay disponibles. Para este proyecto se ha utilizado la versión 1.44, que se puede descargar del siguiente enlace directo:

http://www.airspayce.com/mikem/bcm2835/bcm2835-1.44.tar.gz (Último acceso: Junio 2015)

Una vez descargados, se abre la terminal y se escriben las siguientes instrucciones:

$ cd (la dirección donde esté el archivo descargado)

$ tar zxvf bcm2835-1.44.tar.gz *#Esto descomprimirá el archivo*

$ cd bcm2835-1.xx *#Cambiará al directorio descomprimido*

$ ./configure *#Acciones necesarias para instalar los drivers*

$ make

$ sudo make check

$ sudo make install

Tras seguir estos pasos, los drivers están listos para usarlos.

Para usar sus librerías, hay que compilar, junto con el proyecto, los archivos de código fuente bcm2835.c y bcm2835.h, ya que se utilizan sus funciones.

Para compilar el código hay que incluir el archivo bcm2835.c en la compilación del siguiente modo:

$ gcc nombre_archivo.c bcm2835.c –o *#nombre_ejecutable*

Si se está utilizando la Raspberry Pi 2 hay que realizar una modificación en el archivo bcm2835.h

La Raspberry Pi 2 utiliza otro microprocesador, el bcm2836, por lo que sus direcciones de memoria cambian con respecto al modelo de Raspberry anterior.

Para solucionar esto, la línea

```
#define BCM2835_PERI_BASE               0x20000000
```

debe cambiarse por

```
#define BCM2835_PERI_BASE               0x3F000000
```

De este modo los drivers funcionarán correctamente en la Raspberry Pi 2.

Anexo II

Configuración de la velocidad del bus CAN

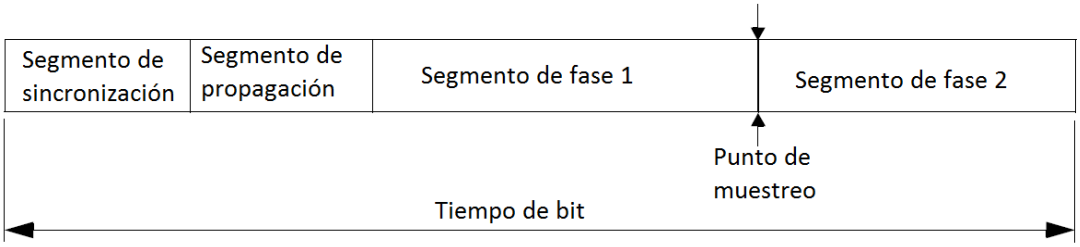
Para configurar la velocidad a la que va a leer la placa CANdiy-shield del bus CAN, hay que modificar una serie de registros de la memoria del microchip MCP2515.

Esta configuración no es trivial, y existen distintas configuraciones teóricamente válidas para cada velocidad, que hay que probar hasta dar con la que funcione correctamente.

Para entender el procedimiento de configuración, hay que definir primero una serie de variables que se van a utilizar

Cada bit que se envía, permanece activo un tiempo determinado, que depende de la frecuencia a la que se está transmitiendo. Este tiempo se llama tiempo de bit (t_{bit})

Cada tiempo de bit, se divide en diferentes segmentos:



Segmento de sincronización: Indica el tiempo que tarda en sincronizarse un bit enviado con todos los nodos. Tiene asociado un tiempo que se denomina $t_{SyncSeg}$.

Segmento de propagación: Recoge el tiempo que tarda la señal en propagarse por todo el bus, y tiene en cuenta los retardos que pueda haber por rebotes de la señal. Tiene asociado un tiempo que se denomina $t_{PropSeg}$.

Los segmentos de fase 1 y 2 (PS1 y PS2): Determinan el punto de muestreo, que se encuentra entre ellos. El punto de muestreo se puede modificar acortando un segmento y alargando otro ya que los tiempos de sincronización y propagación suelen ser fijos. Tienen asociado un tiempo denominado t_{PS1} y t_{PS2}.

De esta forma $t_{bit} = t_{SyncSeg} + t_{PropSeg} + t_{PS1} + t_{PS2}$.

Además se define otro parámetro llamado SJW (Amplitud del salto de configuración), que ajusta el bit del reloj para mantenerlo sincronizado con el mensaje transmitido. Su valor debe ser más grande cuanto más inestable sea el reloj.

Estos tiempos descritos deben ser definidos con respecto al periodo de un oscilador de cuarzo.

Una vez explicados estos conceptos, se procede a configurar la velocidad:

En primer lugar hay que determinar un preescalador (llamado BRP) del oscilador de cuarzo ya que en general su periodo es mucho más pequeño que el de un bit de bus CAN. Este nuevo periodo, que es un múltiplo del periodo anterior, se denomina TQ.

$$TQ = T_{osc} * BRP * 2$$

Se recomienda tomar un valor de BRP de tal forma que t_{bit} sea al menos 16 veces TQ.

A continuación, hay que elegir los valores de los demás tiempos ($t_{SyncSeg}$ $t_{PropSeg}$ t_{PS1} t_{PS2}), en función del valor TQ.

En las hojas de características del microchip MCP2515 se indican unos valores recomendados para un t_{bit} = 16 veces TQ

- SyncSeg = 1 TQ
- PropSeg = 2 TQ
- PS1 = 7 TQ
- PS2 = 6 TQ
- SJW = 1 TQ

Aunque estos son los valores típicos de un bus CAN cualquiera, estos parámetros pueden no funcionar y se deben especificar otros valores.

A la hora de determinar los valores, se deben tener en cuenta las siguientes restricciones, que vienen impuestas en la hoja de características:

$$PropSeg + PS1 \geq PS2$$

$$PropSeg + PS1 \leq T_{Delay}$$

$PS2 > SJW \quad (1 \leq \text{SJW} \leq 4)$

Una vez decididos los valores de cada parámetro, se deben escribir en la memoria del microchip MCP2515.